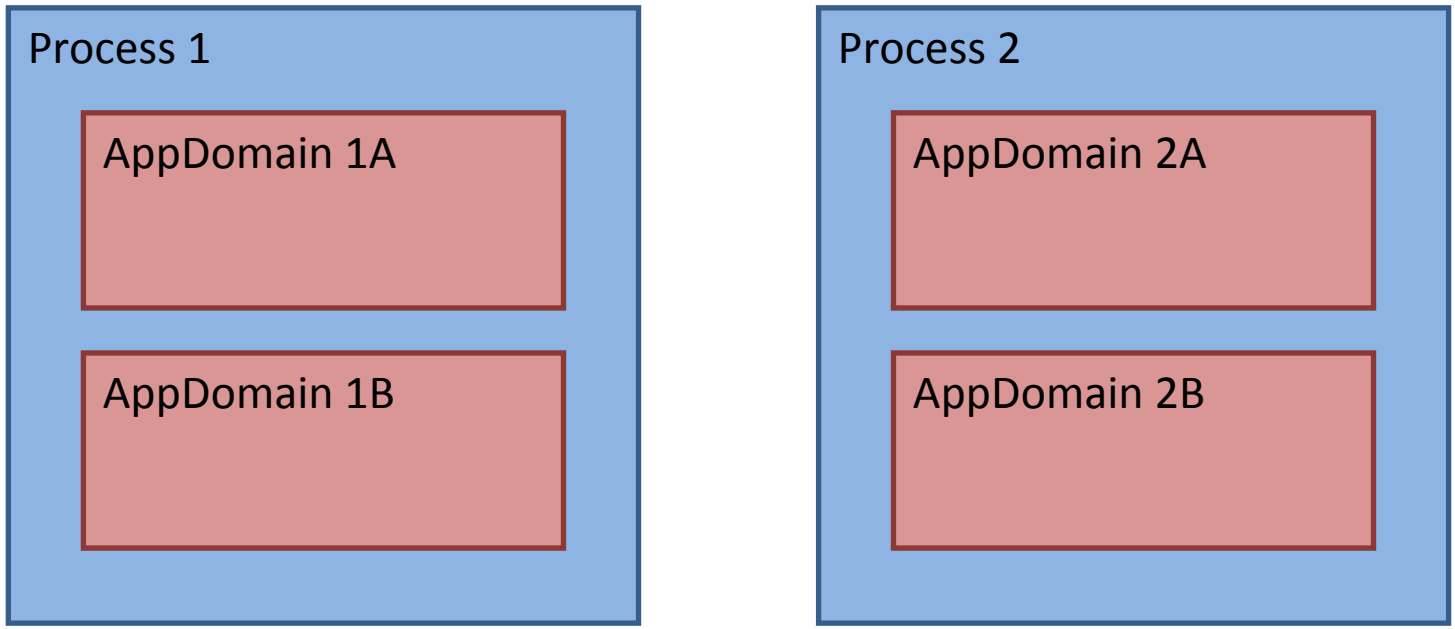
Microsoft .NET Remoting

Morten Dahl
dahl@cs.aau.dk

Application Domains

- Traditionally processes are used to isolate applications
- .NET type system guarantees well-behaved code
  - applications cannot interfere with each other
- AppDomains instead of processes
- Less performance cost (cross-process calls, switching)

Application Domains



Cross-AppDomain Communication

.NET Remoting

- Client-server model
- Abstraction of network
- Object oriented interface
- Distributed garbage collection
- Extendable subsystem
- Server hosting: stand-alone, IIS, or Windows Service
- More a protocol than a service layer (no transactions etc.)

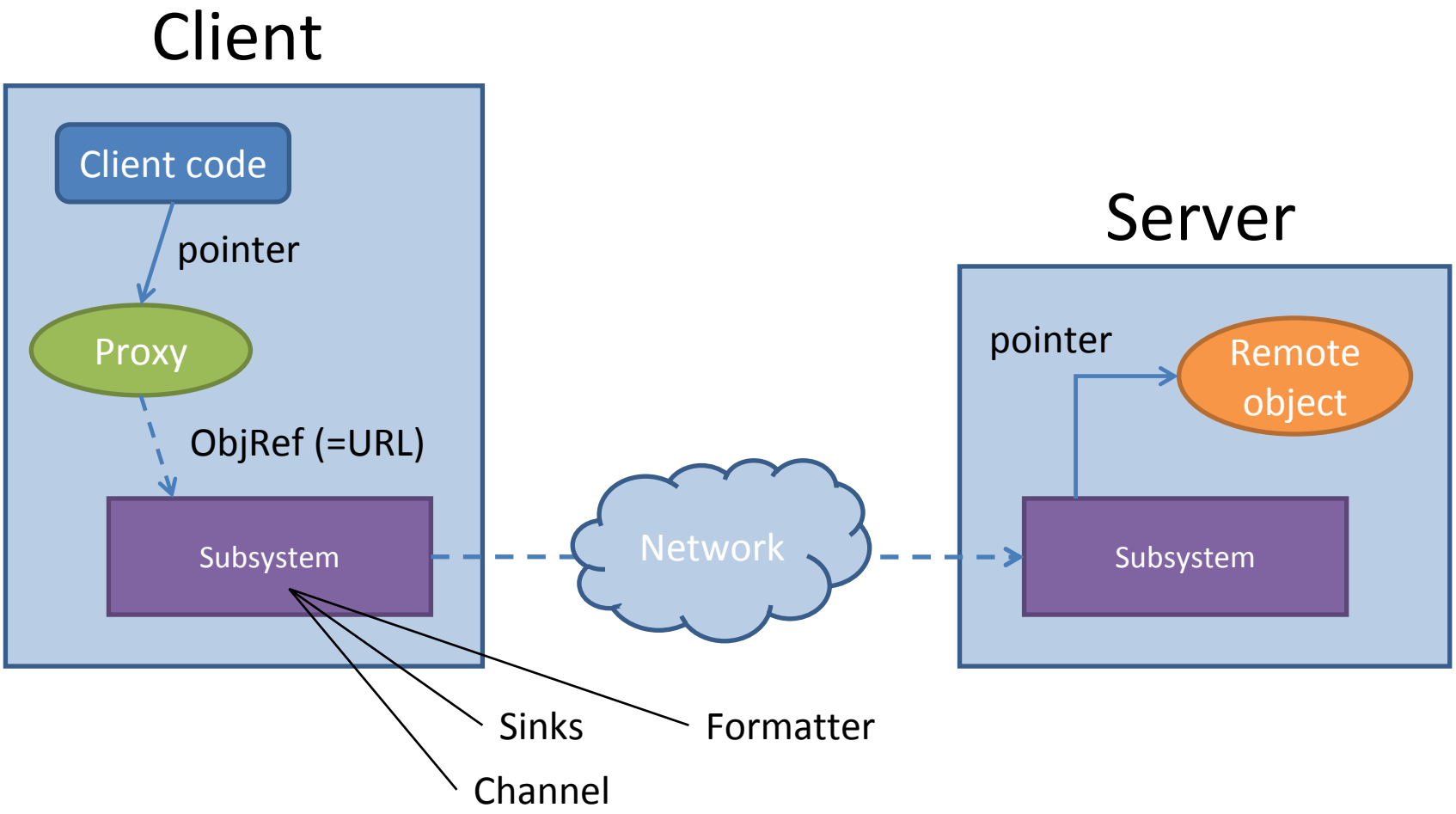
Best practice recommends using Remoting for

- Cross-AppDomain and Cross-Process
- LAN applications

ByValue and Remote Objects

- ByValue (pass by value) objects
  - object is copied across AppDomain boundaries
  - no relationship to original object
  - method calls executed locally
  - must be serialisable: implement *ISerializable* interface or marked with [*serializable*] attribute
- Remote (pass by reference) objects
  - object lives at origin
  - method calls executed at origin
  - referenced by “networked pointer” (*ObjRef*)
  - accessed through transparent proxy
  - must extend *MarshalByRefObject*

Remoting Overview



Kinds of Remote Objects

- Server-activated (SA)
  - not assigned to specific client
  - Singleton: framework guarantees that exactly one copy of object is exposed
    - useful for sharing data between clients
  - SingleCall: framework creates a new object for each method call (destroyed on return)
    - scalable since no state is stored
  - Published: server program creates object and exposes it

- Client-activated (CA)
  - created on request by client
  - each object has its own state (need for object id in ObjRef)
  - similar to local objects only stored at server

Networked Pointers: *ObjRefs*

- Object pointer suitable for network
  - if passed from one client to another it points directly to server and not through first client
  - serialisable
- Server-activated object
  - Server URL: http://server:8000/RemoteObj.rem
- Client-activated object
  - Server URL: http://server:8000/RemoteObj.rem
  - Object ID (GUID)

Binding

- First contact by URL
  - http://server:8000/Gatekeeper
  - from then on we can use ObjRefs
- Activator
  - Obj o = (Obj) Activator.GetObject(typeof(Obj), url);
- new operator
  - Obj o = (Obj) new Obj;
  - URL must be bound to type beforehand
  - cannot be used if all we have is interface!

Storage Example

- Illustrating
  - binding
  - exposing remote objects (server)
  - accessing remote objects (client)
  - difference between kinds

Storage Example: shared

Remote storage object

- Holds an integer value
- Getter and setter for value
- Implements MarshalByRefObject
- (Integer type is serialisable)

```
interface IStorage
{
    void SetValue(int Value);
    int GetValue();
}
```

```
class Storage :
    MarshalByRefObject,
    IStorage
{
    private int value;

    public void SetValue(int Value)
    { this.value = Value; }

    public int GetValue()
    { return this.value; }
}
```

Storage Example: server (SA)

```
HttpChannel channel = new HttpChannel(8000);
ChannelServices.RegisterChannel(channel, false);

RemotingConfiguration
    .RegisterWellKnownServiceType(
        typeof(Storage),
        “Storage.rem”,
        WellKnownObjectMode.Singleton);
```

Storage Example: server (SA)

```
HttpChannel channel = new HttpChannel(8000);
ChannelServices.RegisterChannel(channel, false);

RemotingConfiguration
    .RegisterWellKnownServiceType(
        typeof(Storage),
        “Storage.rem”,
        WellKnownObjectMode.SingleCall);
```

Storage Example: server (SA)

```
HttpChannel channel = new HttpChannel(8000);
ChannelServices.RegisterChannel(channel, false);

Storage store = new Storage();
store.SetValue(42);

RemotingServices.Marshal(
    store,
    “Storage.rem”);
```

Storage Example: client (SA)

```
HttpChannel channel = new HttpChannel();
ChannelServices.RegisterChannel(channel, false);

IStorage store = (IStorage) Activator.GetObject(
    typeof(IStorage),
    “http://server:8000/Storage.rem”);

// use ‘store’ as though local object
```

Storage Example: server (CA)

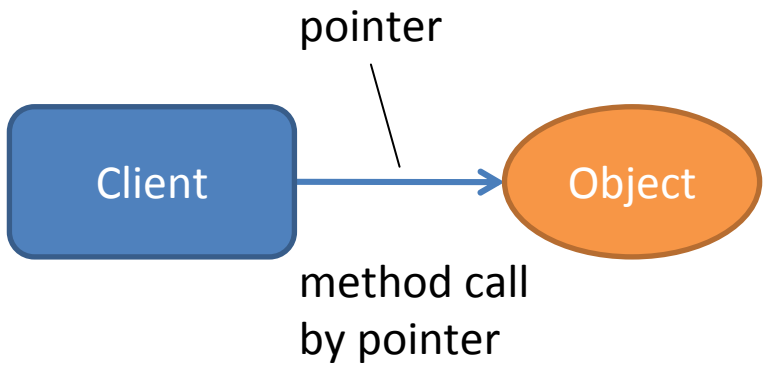
```
HttpChannel channel = new HttpChannel(8000);
ChannelServices.RegisterChannel(channel, false);

RemotingConfiguration.ApplicationName =
    "Storage.rem";

RemotingConfiguration.
    RegisterActivatedServiceType(
        typeof(Storage));
```

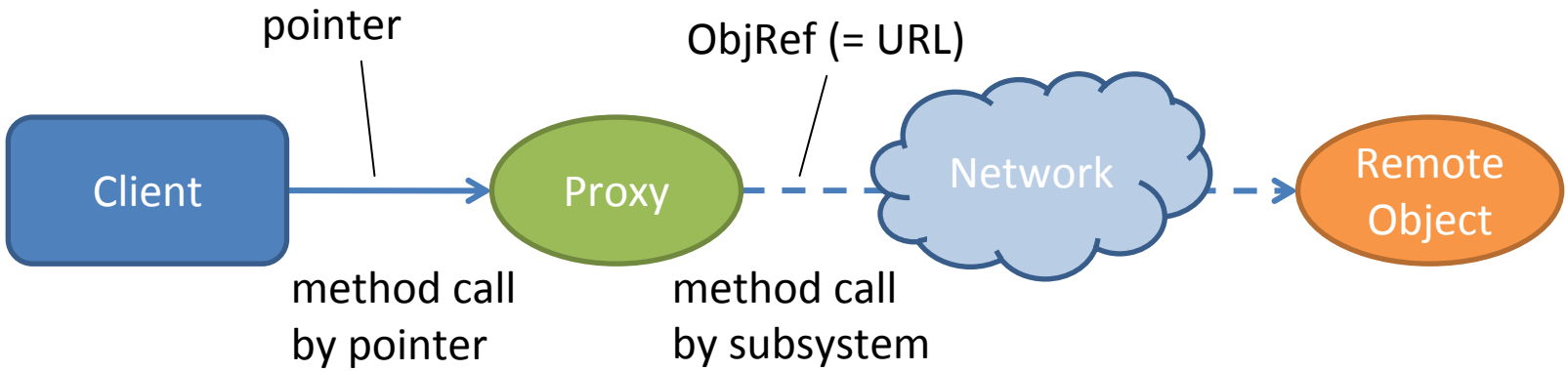
Storage Example: client (CA)

```
HttpChannel channel = new HttpChannel();
ChannelServices.RegisterChannel(channel, false);

RemotingConfiguration.
    RegisterActivatedClientType(
        typeof(Storage),
        “http://server:8000/Storage.rem”);

Storage storeA = (Storage) Activator.CreateInstance(typeof(Storage));
Storage storeB = new Storage();

// use ‘storeA’ and ‘storeB’ as though local objects
```

Break

Method Calls

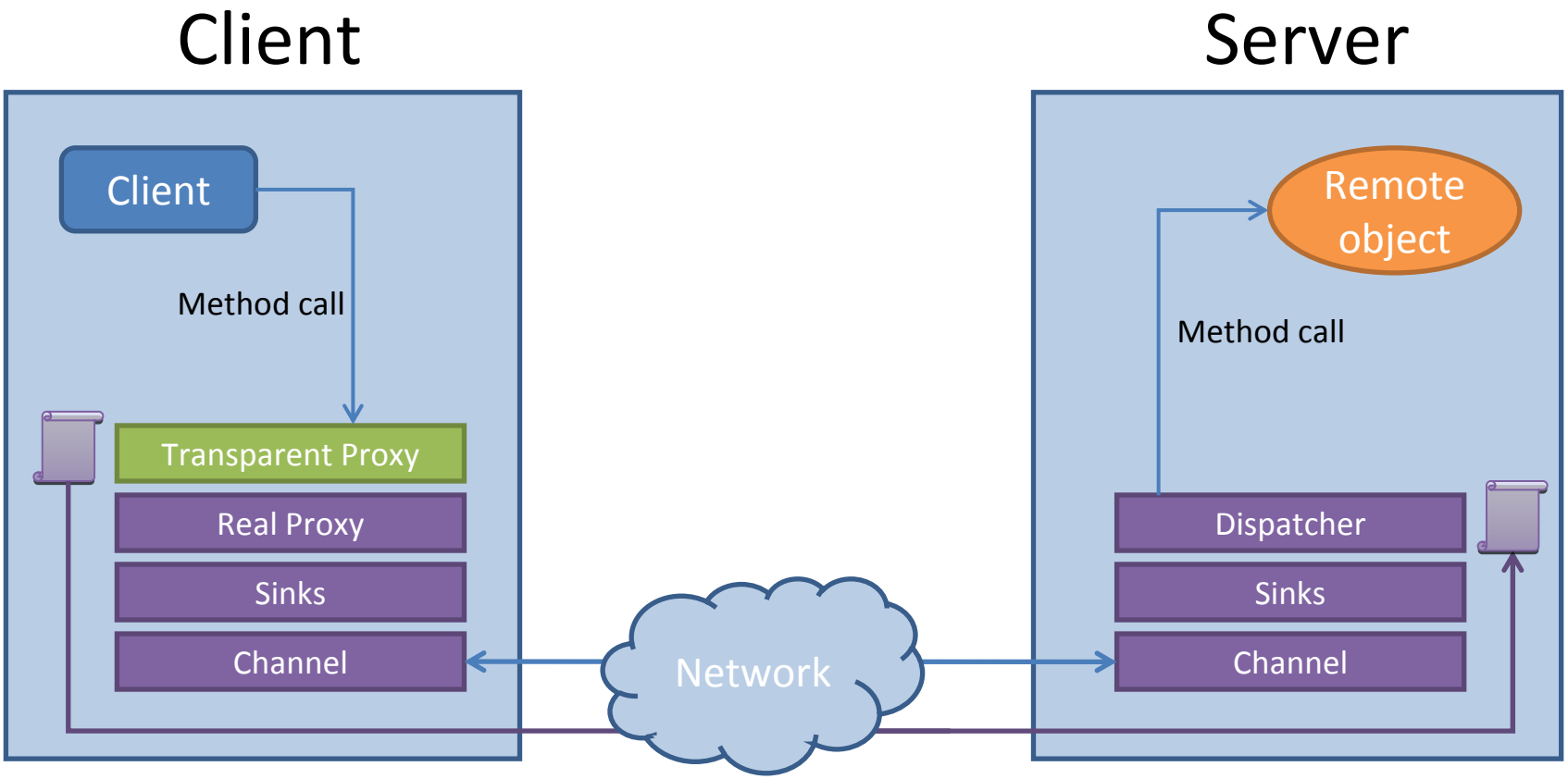




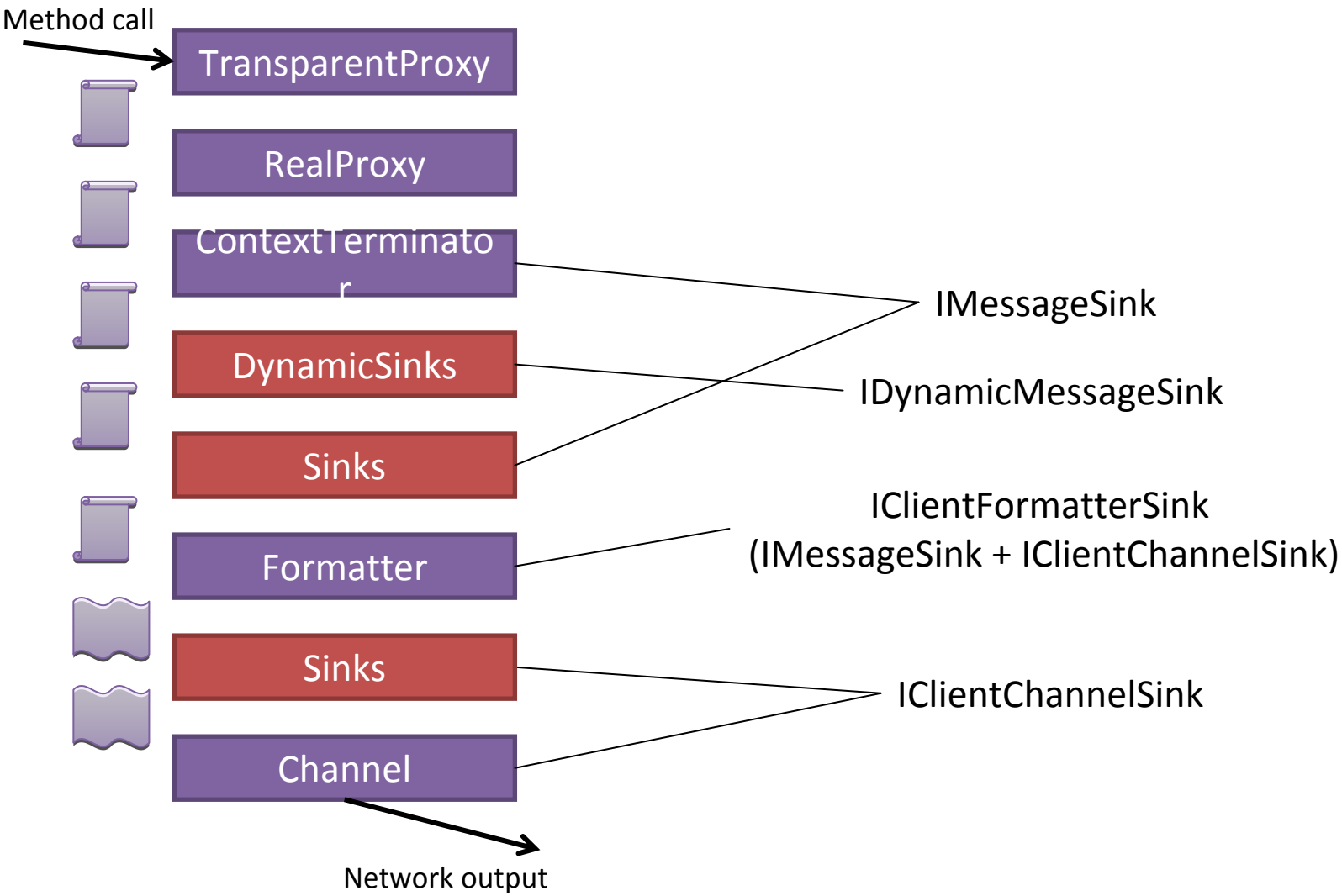
Remote Method Call

- Client calls method on proxy with same interface; proxy transfers call to server
- Input and output parameters must be ByValue or remote objects
  - client can potentially become server
- Synchronously, asynchronously, or one-way asynchronously (fire-and-forget; no return value nor exceptions)

Remote Method Call



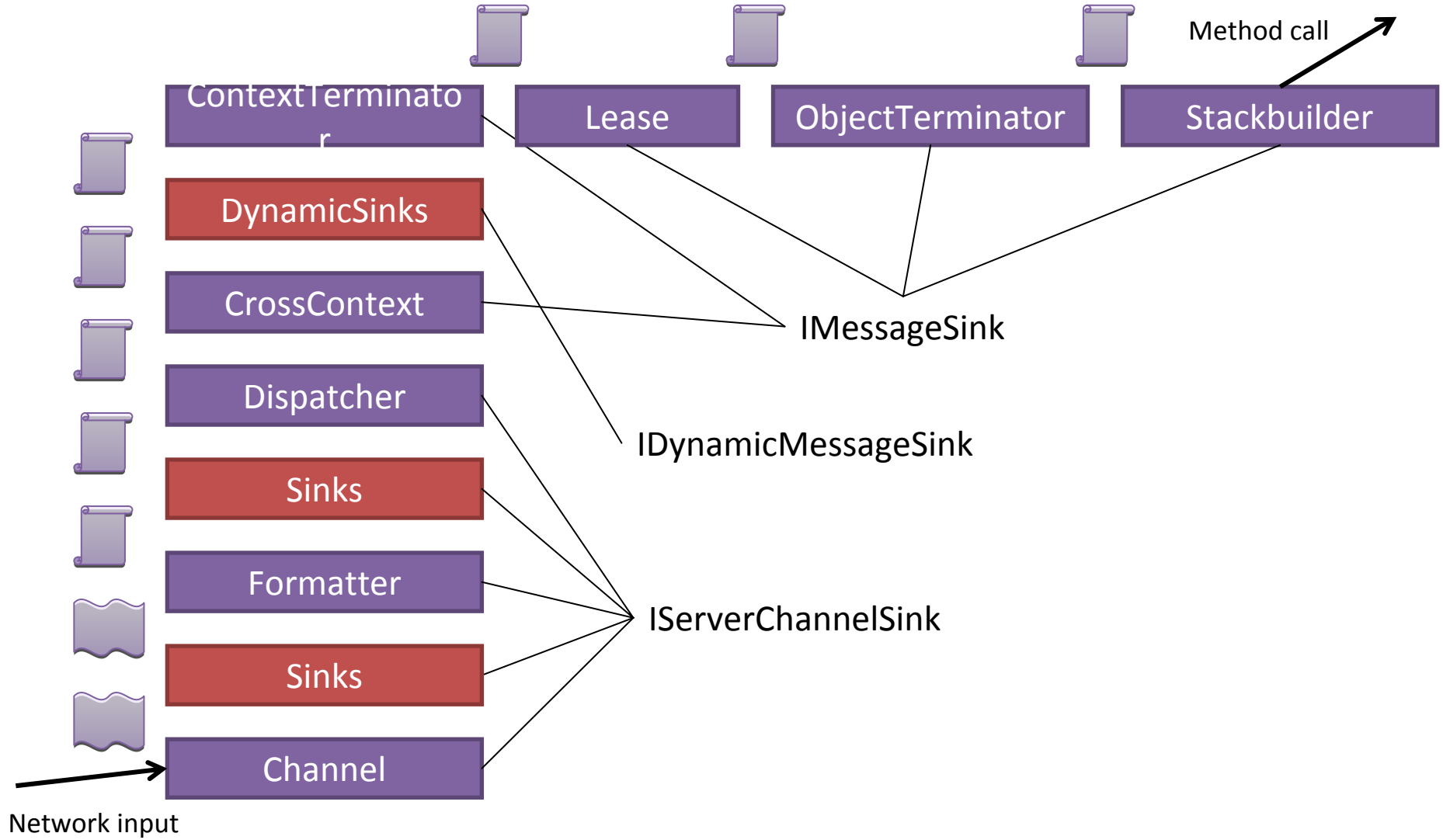
Client-side Messaging



Client-side Messaging

- Transparent Proxy: same interface as remote object; converts between method call and message
- Message: represents method call (and parameters) via dictionary
- Real Proxy: takes message and forwards it to sinks; holds ObjRef
- Context Terminator/Dynamic Sinks: Context Terminator notifies dynamic sinks associated with context of method call
- Sinks: optional message processing
  - e.g. thread priority
- Formatter: serialisation of message into stream
- Sinks: optional stream processing
  - e.g. compression
- Channel: implements network protocol (TCP, HTTP, custom)

Server-side Messaging



Server-side Messaging

- Channel: implements network protocol (TCP, HTTP, custom)
- Sinks: optional stream processing
  - e.g. decompression
- Formatter: deserialisation of stream into message
- Sinks: optional message processing
  - e.g. thread priority
- Dispatcher: checks for disconnected or timed-out objects; creates SAO objects
- Cross Context/Dynamic Sinks: notifies dynamic sinks
- Context Terminator/Lease: calls *RenewOnCall* of target object's lease
- Object Terminator/Stackbuilder: constructs stack frame

Built-in Components

- Formatters
  - BinaryFormatter
  - (SoapFormatter)
- Channels
  - TcpChannel (authentication and encryption)
  - HttpChannel (authentication and encryption in IIS)
  - IpcChannel (authentication)
- Custom formatters and channels

Sink Example

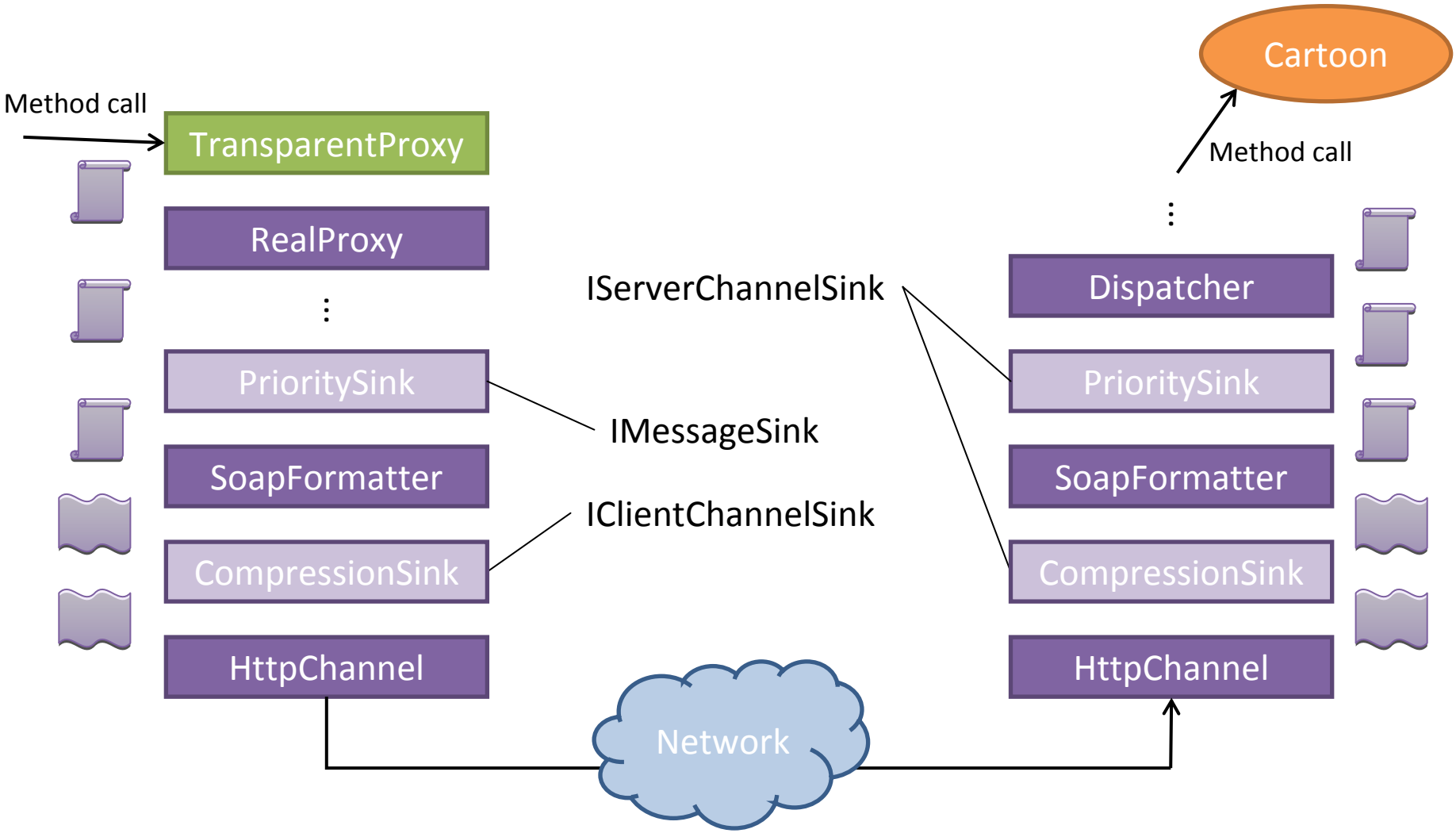
- Illustrating
  - messaging subsystem
    - SoapFormatter + HttpChannel
    - compression channel sink
    - thread priority message sink
  - configuration files

Sink Example: shared

- Cartoon remote object
  - Name property
  - Behave method
    - one-way as we suspect it having no effect what-so-ever
  - Sleep
    - for when behaving becomes tiresome

```
public class Cartoon : MarshalByRefObject
{
    private string name;
    public string Name
    { get {return name; }
      set {name = value; } }

    [OneWay]
    public void Behave()
    { … }

    public int Sleep()
    {
        int napDuration = 4000;
        Thread.Sleep(napDuration);
        return napDuration;
    }
}
```

Sink Example: subsystem



Sink Example: basic config

- HttpChannel channel = new HttpChannel();
  ChannelServices.RegisterChannel(channel, false);
- Cartoon loony = (Cartoon)Activator.GetObject(typeof(Cartoon),
  "http://localhost:8000/Cartoon.rem");

- <?xml version="1.0" encoding="utf-8" ?>
  <configuration>
  <system.runtime.remoting>
  <application>
  <channels>
  <channel ref="http"></channel>
  </channels>
  <client>
  <wellknown type="Shared.Cartoon, Shared“
  url="http://localhost:8000/Cartoon.rem" />
  </client>
  </application>
  </system.runtime.remoting>
  </configuration>
- **Cartoon loony = new Cartoon();**

Sink Example: full config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <channels>
        <channel ref="http">
          <clientProviders>
            <provider type="Remoting.PriorityClientSinkProvider, Remoting" />
            <formatter ref="soap" />
            <provider type="Remoting.CompressionClientSinkProvider, Remoting" />
          </clientProviders>
        </channel>
      </channels>
      <client>
        <wellknown type="Shared.Cartoon, Shared" url="http://localhost:8000/Cartoon.rem" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Sink Example: sinks

- Compression channel sink
  - compresses stream before send on channel
  - decompresses stream after received on channel

```
public void ProcessMessage(IMessage msg,
      ITransportHeaders reqHeaders, …)
{
1.  requestStream =
      GetCompressedStream(requestStream);

2.  nextSink.ProcessMessage(msg, requestHeaders,
      …);

3.  responseStream =
      GetDecompressedStream(responseStream);
}
```

- Priority message sink
  - stores client thread priority in message
  - gives server thread priority from message

```
public IMessage SyncProcessMessage(IMessage
      msg)
{
1.  LogicalCallContext lcc =
      msg.Properties["__CallContext"];

2.  lcc.SetData("Priority",
      Thread.CurrentThread.Priority);

3.  return nextSink.SyncProcessMessage(msg);
}
```

Break

Distributed Garbage Collection

- Error occurs if client accesses collected object; when to garbage collect?
- Reference counting
  - client can crash
- Alive!-pings
  - not suitable in all scenarios: firewalls, network load
- Remoting: Leases and Sponsors
  - lease keeps object alive for a period; sponsor can extend this period
  - more general: can implement alive!-pings as well as other approaches

Leases and Sponsors

- Leases
  - each object gets lease on creation
  - object not collected until lease expires
  - time-to-live counter
    - initial time; 5 min by default
    - extension on call; 2 min by default
    - sponsor contacted when reaches zero
- Sponsors
  - must decide if object's lease should be extended
  - can live on client; time-out handles client crash

Making Objects Live Longer – Leases

- Object-wide

```
class LongerLiving : MarshalByRefObject
{
    public override object InitializeLifetimeService()
    {
      ILease lease = (ILease) base.InitializeLifetimeService();
      if (lease.CurrentState == LeaseState.Initial)
      {
                lease.InitialLeaseTime = TimeSpan.FromMinutes(10);
                lease.RenewOnCallTime = TimeSpan.FromMinutes(5);
      }
      return lease;
    }
}
```

Making Objects Live Longer – Leases

- Object-wide

```
class InfinitelyLiving : MarshalByRefObject
{
    public override object InitializeLifetimeService()
    { return null; }
}
```

- Application-wide

```
LifetimeServices.LeaseTime = TimeSpan.FromMinutes(10);
LifetimeServices.RenewOnCallTime = ..
LifetimeServices.SponsorshipTimeout = ..
```

  – can be set in configuration file

Making Objects Live Longer – Sponsors

```
class MySponsor :
    MarshalByRefObject,
    ISponsor {

public TimeSpan Renewal(ILease lease)
{
    if (needsRenewal())
    { return TimeSpan.FromMinutes(5); }
    else
    { return TimeSpan.Zero; }
}
}

private bool needsRenewal() { ... }
```

```
ILease objLease = (ILease) obj.GetLifetimeServices();
MySponsor sponsor = new MySponsor();
objLease.Register(sponsor);
```

Keep Alive Sponsor

```
class KeepAliveSponsor :
      MarshalByRefObject,
      ISponsor {
private DateTime lastKeepAlive;
public KeepAliveSponsor()
{ lastKeepAlive = DateTime.Now; }

public void KeepAlive()
{ lastKeepAlive = DateTime.Now; }

public TimeSpan Renewal(ILease lease)
{
      TimeSpan duration = diff(DateTime.Now);
      if (duration.TotalSeconds < 5)
      { return TimeSpan.FromSeconds(10); }
      else
      { return TimeSpan.Zero; }
}
}
```

```
KeepAliveSponsor sp = createSponsorOnServer();

Pinger pinger = new Pinger(sp);
pinger.Start();

ILease lease = (ILease) obj.GetLifetimeService();
lease.Register(sp);

...

lease.Unregister(sp);
pinger.Stop();
```

Code Distribution

Assembly Loading

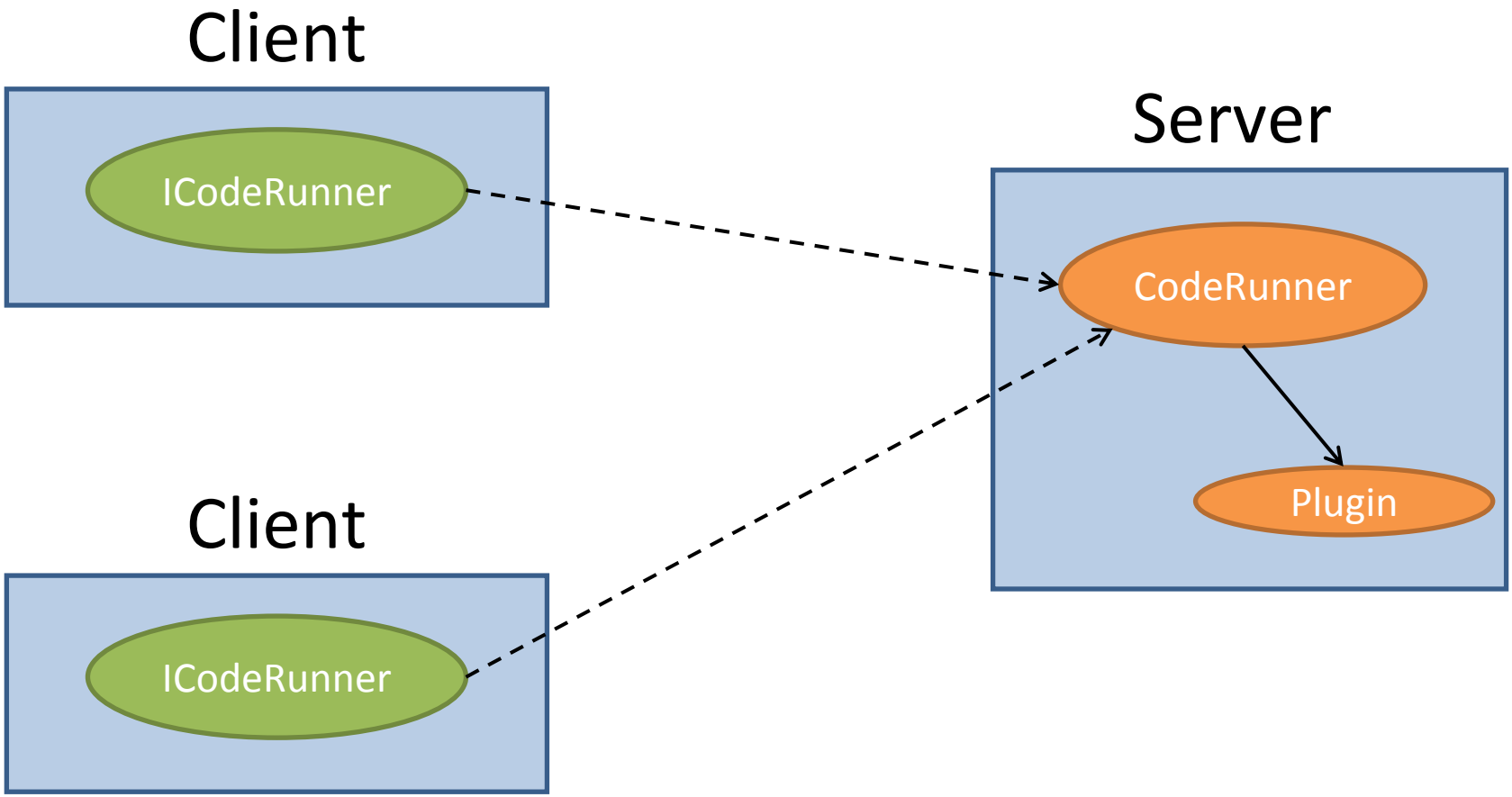
1. determine correct assembly to load (configuration files, publisher policy file)
2. cache and GAC
3. probing
   - codebase configuration
   - heuristic probing of paths (.\bin, .\[culture]\bin, ...)

```
<configuration><runtime><assemblyBinding>
  <dependentAssembly>
    <assemblyIdentity name= "MyAssembly"/>
    <codeBase version="2.0.0.0"
       href="http://www.server.com/MyAssembly.dll"/>
  </dependentAssembly>
</assemblyBinding></runtime></configuration>
```

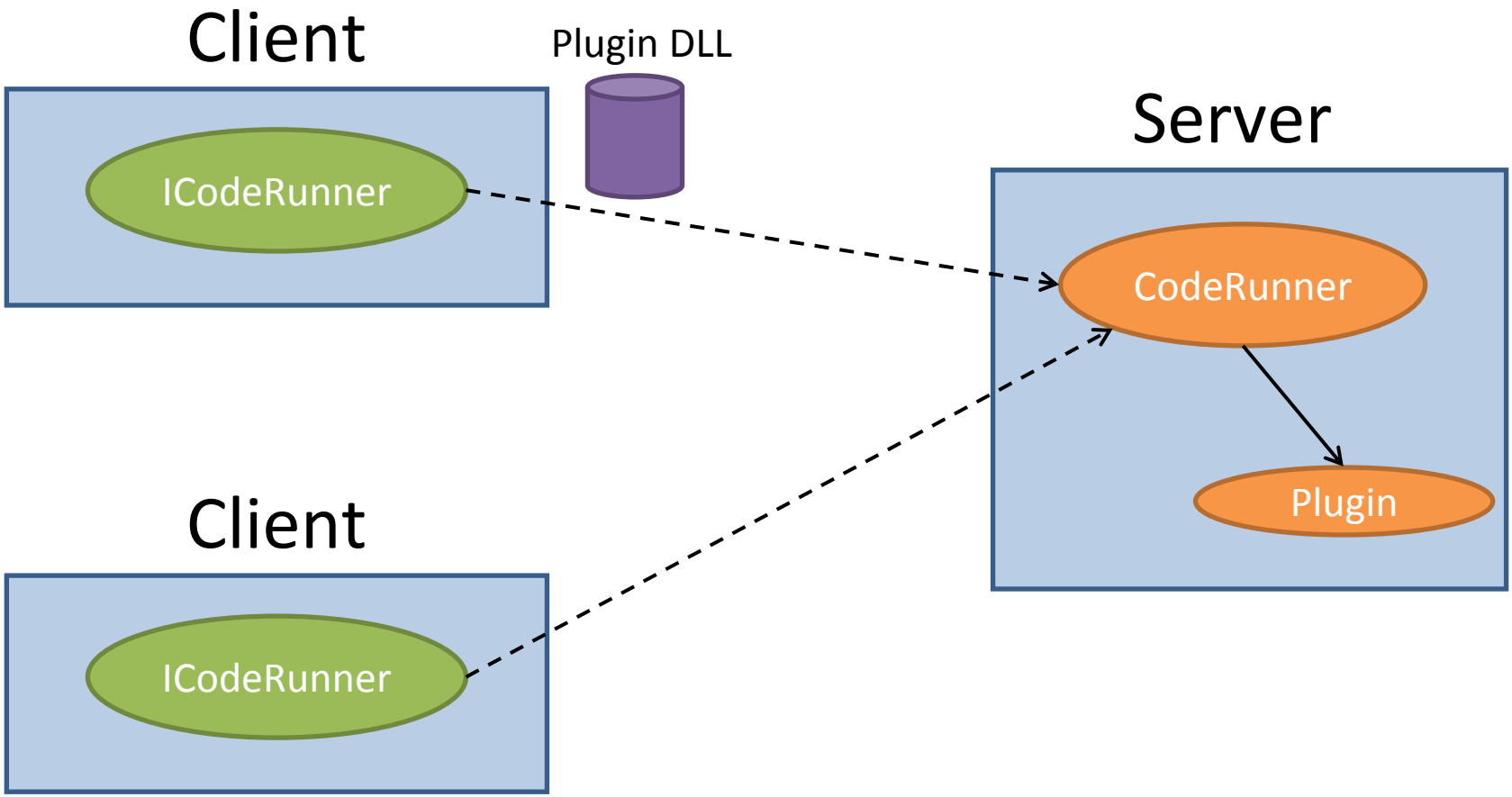
Code Runner Example

- Illustrating
  - easy way to have fun

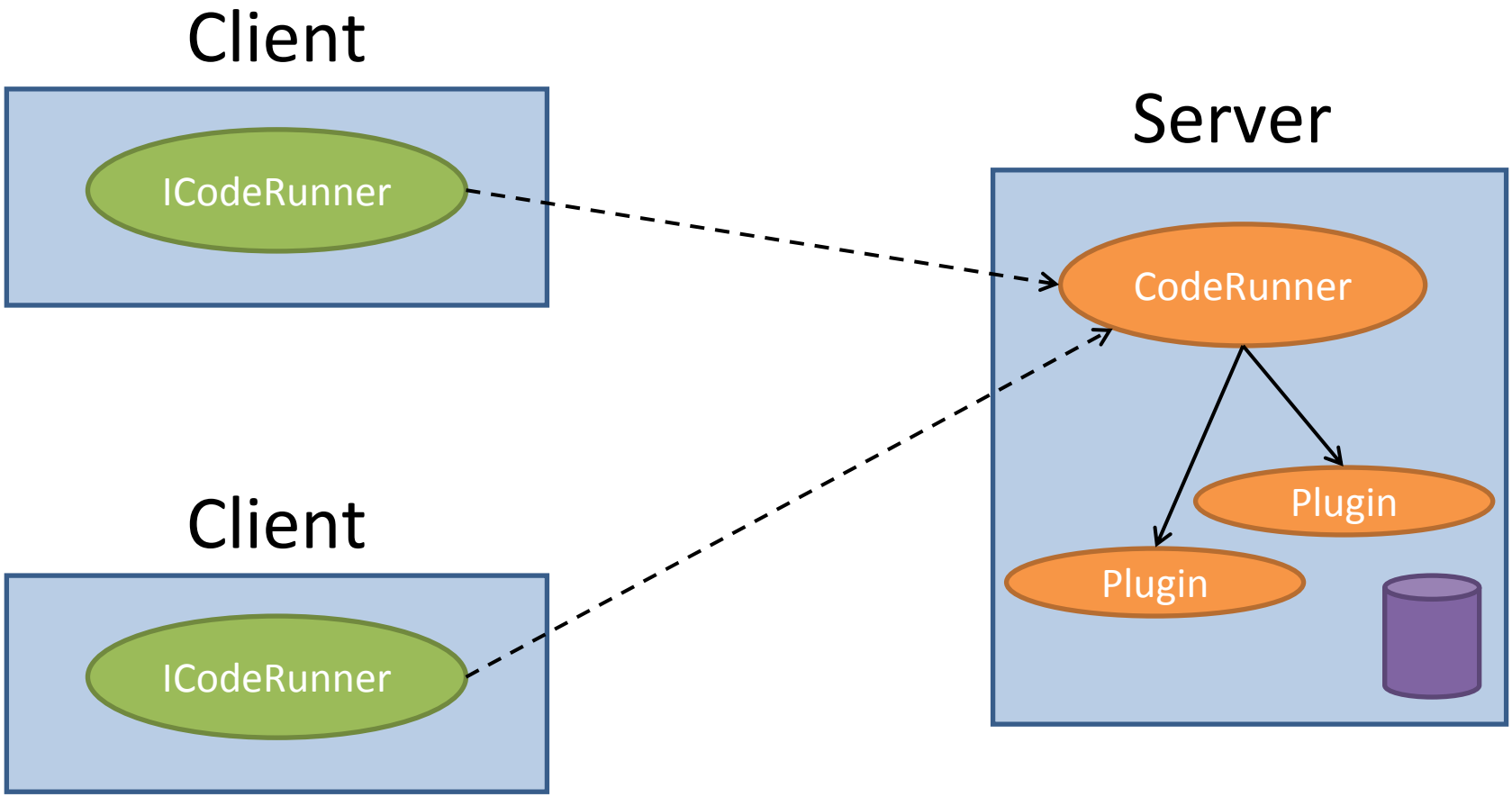
Code Runner Example: overview



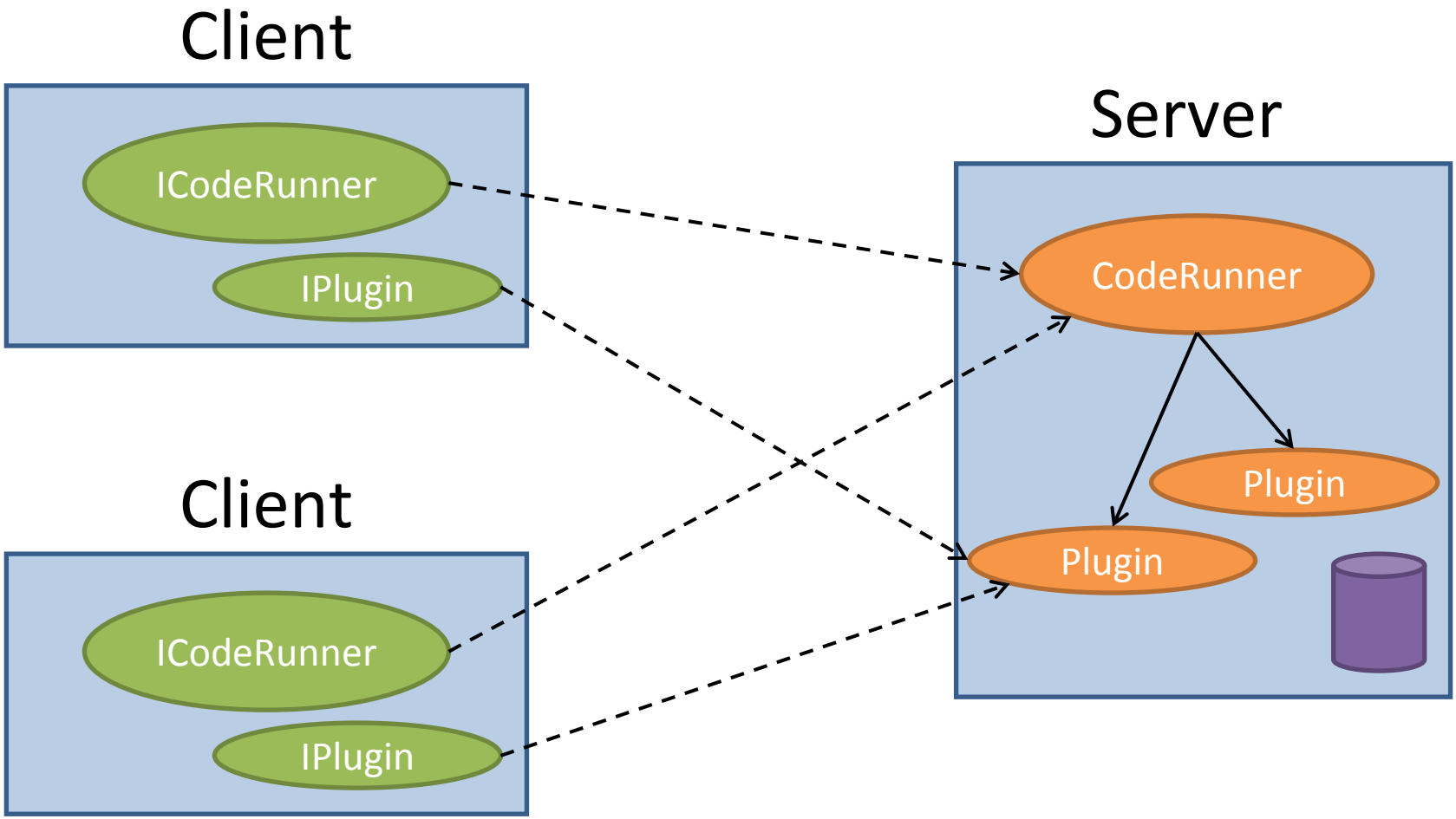
Code Runner Example: overview



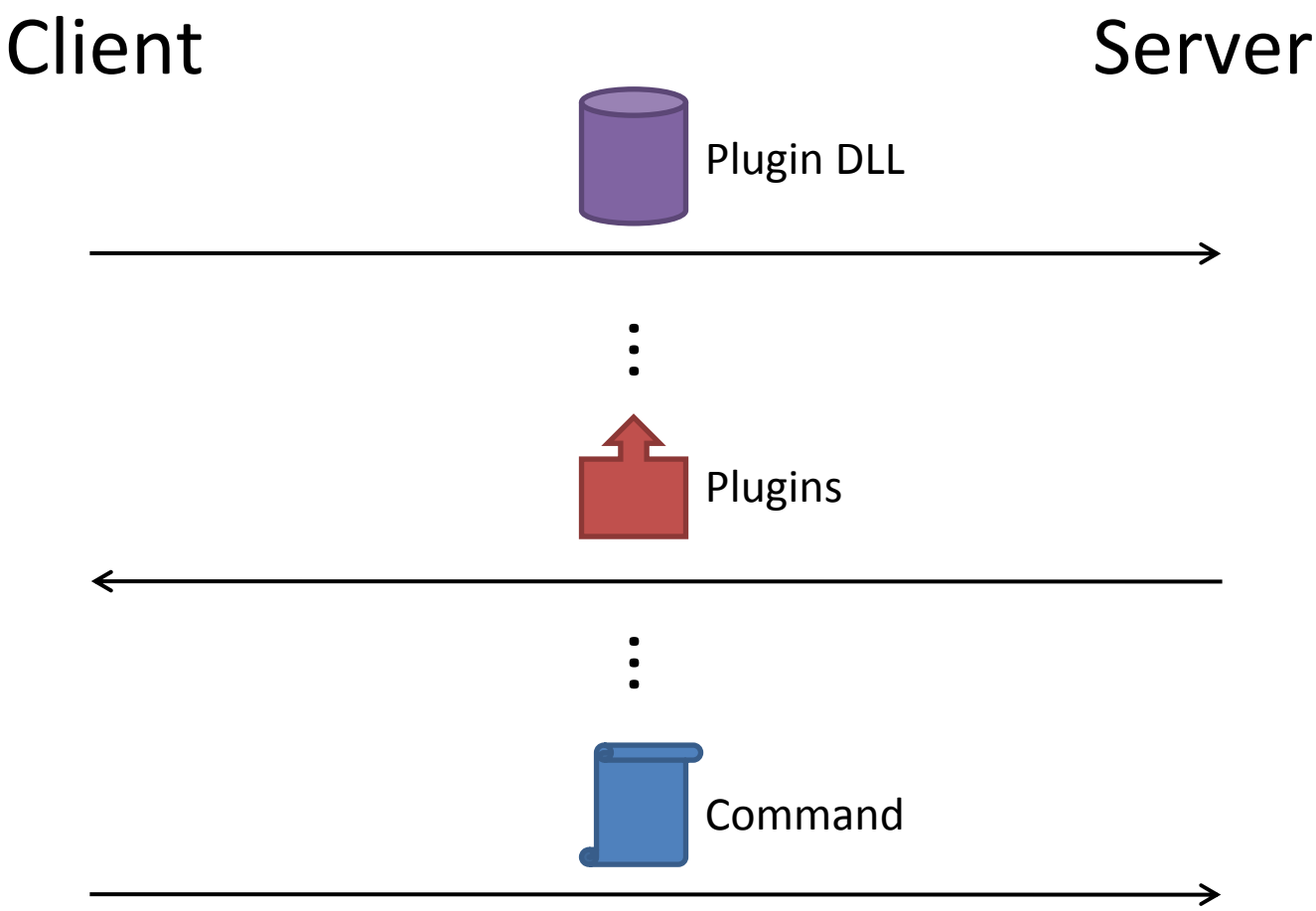
Code Runner Example: overview



Code Runner Example: overview

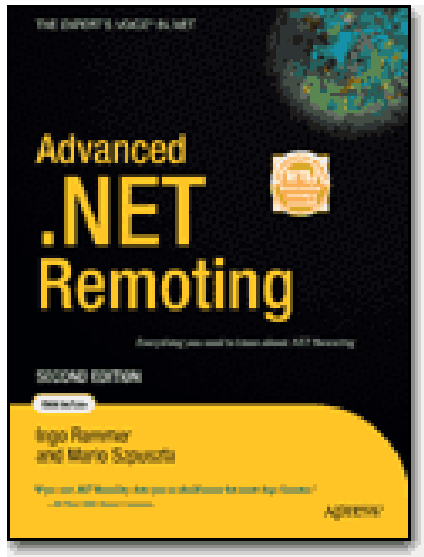


Code Runner Example: interaction



References

- Book: Advanced .NET Remoting, 2nd Edition
  - Ingo Rammer, Mario Szpuszta
  - ISBN 1590594177
  - Apress 2005
- .NET Remoting Overview
  - MSDN Library
- Google



That leaves EXPERIENCED NET REMOTING Developers. Note I mentioned both "EXPERIENCED", "NET" and "REMOTING", though I doubt there would be use for those folks either. Cannot explain why this book would get ONE positive comment.Â ".NET Remoting" by McLean, Naftel and Williams is by far the best. What I like most about it is the writing style; it is clear and concise. What's more, the writing is grammatical and at times best described as elegant. .NET Remoting Internet Research Medical Records Research Data Encoding Microsoft PowerPoint Virtual Assistant Microsoft Excel PowerPivot Bookkeeping Data Entry Microsoft Excel. Overview. I have a bachelor's degree in Biology major in Human Biology from De La Salle University in 2010. I have developed skills on research and information gathering that would get the work done in the shortest possible time. I also have wide experience using MS office, PDF and data encoding. I'm a fast learner. Microsoft .NET Remoting provides a framework that allows objects to interact with each other across application domains. Remoting was designed in such a way that it hides the most difficult aspects like managing connections, marshaling data, and reading and writing XML and SOAP. The framework provides a number of services, including object activation and object lifetime support, as well as communication channels which are responsible for transporting messages to and from remote applications.